# FastID: an Undeceived Router for Real-time Identification of WiFi Terminals

Li Lu*, Runzhe Wang*, Jing Ding*†, Wubin Mao*, Wei Chen* and Hongzi Zhu†
*School of Computer Science and Engineering, University of Electronic Science and Technology of China
†Department of Computer Science and Engineering, Shanghai Jiao Tong University, China

***Abstract*—In recent past, the rapid developing of mobile internet inspires the widespread use of WiFi (IEEE 802.11) technology. In WiFi, the access control of a terminal to the router remains a significant challenge because the PIN (password) and MAC address are easy to guess and forge. In this paper, we present FastID - a practical system that identifies WiFi terminals in real-time by fingerprinting their clocks. Previous approaches of clock fingerprinting require tens of minutes or even hours for clock data collection, and thus cannot be applied into real-time WiFi terminal identification. Even worse, unstable wireless communications and unknown status of terminals' OSes may further degrade the accuracy of fingerprint computation. In comparison, FastID performs fast clock fingerprinting based on the timestamps carried by terminals' ICMP packets. Moreover, FastID employs simple but efficient techniques to remove outliers of collected clock data and differentiate terminals based on the similarity of their distributions, making it suitable for fast terminals identification. FastID is implemented on an off-the-shelf commercial WiFi router and extensively evaluated based on 10 commodity WiFi terminals. Experimental results show that FastID is able to identify terminals with high accuracy and low cost within several seconds.**

***Index Terms*—WiFi terminal; Identification; Clock skew.**

## I. Introduction

With the fast developing of mobile internet, the technology of WiFi becomes one of most important technologies of internet access. However, due to the importance in providing ubiquitous services and the inherent vulnerability of the broadcast nature of the wireless medium, WiFi is becoming the target of a number of attacks. One of the most severe way in which WiFi can be attacked is unauthorized access to WiFi routers (or access points, APs). To defend against this attack, PIN (password) and MAC address binding have been employed to identify WiFi terminals. However, these techniques cannot perfectly protect routers from unauthorized access as PIN and MAC address are easy to guess and forge. For example, attackers can mimic a valid router and inveigle authorized WiFi terminals to input their PINs and capture their MAC addresses. Moreover, the access control to a router requires fast terminal identification which means the router must identify terminals within at most several seconds.

In this paper, we present FastID - a fast and practical system that identifies terminals by inferring their physical clock fingerprints with high accuracy and minimum overhead. This scheme, like the approach proposed by Huang et al., [1] to fingerprint devices in Bluetooth communication, is based on estimating clock skews of wireless terminals in 802.11 communications. Basically, the operation of most digital circuit systems, such as computer systems, is synchronized by a "clock" that dictates the sequence and pacing of the devices on the circuit. The maximum speed at which a system can run must account for the variance that occurs between the various elements of a circuit due to differences in physical composition, temperature, and path length. Thus, clocks on different circuits at the same frequency may drift to each other. The difference between these clock drifts is defined as clock skews.

Previous studies illustrate that clock skew can be used for identifying devices, such as PCs in wired networks [2] and WiFi APs [3]. However, these schemes fail to fingerprint wireless terminals in 802.11 communications as they exclusively rely on the timestamps carried by packet headers to state transmission times, which can be easily delayed by the unstable wireless communications as well as unknown running conditions of operating systems on terminals. Furthermore, these approaches cannot be applied in the real-time circumstance like ours that needs fast fingerprint detection and terminal identification. For example, Kohno et al.'s work [2] needs hours of collecting sufficient data for fingerprinting. Jana and Sneha [3] propose a scheme which identifies WiFi APs in 20 to 30 minutes. Another interesting work, BlueID [1], can fingerprint Bluetooth devices in seconds based on the temporal feature of Bluetooth baseband, but their scheme can only be applied to frequency hopping based wireless systems rather than 802.11 compliant ones.

The proposed FastID employs timestamps carried by ICMP message type 13 [4] packets to compute clock skews of WiFi terminals. Specifically, once a terminal is signing into a router, the router intentionally sends ICMP time requests to the terminal, and calculates the clock skew of the terminal from timestamps in its responses. However, unlike the approaches aforementioned, to realize FastID, there are several practical challenges must be addressed. First, sufficient data for fingerprinting should be collected within very short time, mostly less than several seconds. Thus, in our approach, we exploit ICMP message type 13 to make a router actively probe timestamps of WiFi terminals very rapidly. Second, unstable wireless communications may incur some outliers in timestamps and then degrade the accuracy of the calculated clock skew. To address this issue, we design an efficient preprocessing algorithm to filter outliers in timestamps. Third,

ISBN 978-3-901882-68-5 

unknown status of the operating systems of WiFi terminals may cause unpredictable delay on ICMP packet transmissions, as all packets receiving and sending should be processed in the kernel of OS. For example, the running of high CPU usage process will delay the packet sending on the network driver. To address this issue, we design a novel identification algorithm based on the similarity of clock skew distributions to minimize this impact of unknown OS status.

We implement FastID on a MERCURY MW150R wireless router with OpenWRT [5] - an open source Linux distribution for embedded devices which can be installed on commercial WiFi routers. We extensively evaluate FastID based on 10 terminals including three android smartphones, three tablets and four laptops which have been installed both Ubuntu 12.04 and Windows 8.1 operating systems. We measure the clock skews in three residential settings: corridor, indoor and outdoor. We test the accuracy of FastID under different distance between the router and terminals and find that the accuracy of measured clock skews are independent of distance. Also, we exam the performance of FastID in terms of OS status. The experimental results illustrate that clock skews remain consistent over time on the same WiFi terminal and vary significantly on different devices. These results validate the use of clock skew as a reliable fingerprint for WiFi device identification. Our results show that FastID can effectively identify WiFi terminals within common communication range of WiFi with high accuracy (99%), low delay (less than 6 seconds in identification) and minimum computational overhead.

We highlight our contribution as follows.

- We present a robust and fast approach on WiFi routers to identify terminals with high accuracy by efficient filtering outliers and fingerprinting terminals based on the similarity of clock skew distributions.
- We implement our approach on an off-the-shelf router without any modification on the hardware to illustrate the potential utility on a large scale of current deployed routers.
- We conduct extensive experiments in different settings and comprehensive analysis in terms of security and performance. The results show that our approach can effectively fingerprint and identify terminals with low cost, and is resilient to timestamp spoofing.

The rest of this paper is organized as follows. We present related work in Section II. Section III introduces the background of clock skew and gives the methodology of our approach. We specify the design of FastID in Section IV and its implementation in SectionV. Section VI evaluates the performance of FastID. We discuss the security and some design issues in Section VII. The whole work is concluded in Section VIII.

## II. Related Work

Device fingerprinting serves many legitimate purposes, including mitigating denial-of-service attacks, preventing fraud, protecting against account hijacking, and curbing content scraping, and other automated nuisances. Some researches show that technologies of device fingerprinting can be used to track users. Pang et al., [6] present a mechanism that can identify users based on their patterns of network traffic, such as the distributions of packet length and visited websites. A practical system, named Multi-Layer Device Fingerprinting [7], has been developed to collect a comprehensive set of traffic data that positively identifies a device. However, these traffic based approaches need hours of data collection to achieve satisfactory accuracy and cannot fulfill the requirement of real-time identification.

Some approaches exploit driver level features to fingerprint devices. The drivers of WLAN client can be fingerprinted with the pattern of access point scanning [8], [9]. Similarly, Mirza et al., [10] identifies 802.11 rate adaptation algorithms by analyzing the bit rates of overheard packets. However, these driver based approaches cannot separate devices which have same type of drivers. Furthermore, the features used by these traffic and driver based approaches are not actual fingerprints of devices as they can be easily modified by changing the configuration of devices.

Some approaches fingerprint wireless transmitters based on their signal level features, such as modulation imperfections [11] and transient states of RF [12]. A good but not yet exploited feature - harmonic wave of oscillator, can be used to differentiate wireless devices with pretty high accuracy. These signal level based approaches, however, are infeasible to be implemented on WiFi devices as measuring the signal level features require prohibitively expensive instruments such as Network Analyzer and Spectrum Analyzer.

Clock skew based approaches are cost efficient (i.e., no specialized instruments required) and robust (i.e., exploiting hardware) comparing with approaches mentioned above. Huang et al., [1] present a practical scheme named BlueID that can identify Bluetooth devices based on their clock skews. However, BlueID can only be applied in the communication systems with frequency hopping. Unfortunately, current WiFi standards do not support the frequency hopping due to the limited channel bandwidth. Jana and Sneha [3] exploit the timestamp of periodic beacon to measure the clock skew of WLAN access point. The most similar work to ours is proposed by Kohno et al., [2] to measure the clock skew of a remote PC in wired network based on the observed time drifts in TCP/ICMP timestamps. However, these two approaches passively receive packets from fingerprintee and thus needs long time for data collection that is not suitable for the real-time identification of terminals on wireless router.

Compared with these clock skew based approaches, our use of clock skew to fingerprint a wireless terminal is not new. However, our contribution is also significant because we apply the clock skew based fingerprinting to the real-time scenario where the identifications are much faster and more accurate compared to the aforementioned clock skew based approaches.

## III. Background and Methodology

In this section, we firstly introduce the background of clock skews, then we present the methodology of the calculation of

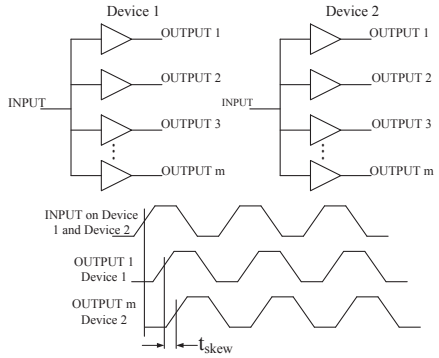



Fig. 1: Device-to-device skew $t_{skew}$ is the time difference of two devices' outputs

clock skews.

### A. Clock Skews

Skew is the time delta between the actual and expected arrival time of a clock signal. Skew can be either extrinsic or intrinsic. The latter is internal to the driver (generator circuitry) and defined as the difference in propagation delays between the device outputs. On the other hand, extrinsic skew is the time difference due to unbalanced trace lengths and/or output loading.

In fact, the clock skew used in our work is the device-to-device skew [13]. This kind of clock skews is defined as the magnitude of the difference in propagation delays between any specified outputs of two separate devices operating at identical conditions, as shown in Fig.1. The devices must have the same input signal, supply voltage, ambient temperature, package, load, environment etc. In our work, we assume that wireless network drivers of both the router and terminals have the same hardware architecture and follow the same IEEE 802.11 standard, and thus, the clock skew of a terminal with respect to the router is the difference in packet sending delays.

### B. Methodology of Clock Skew Calculation

In an IEEE 802.11 infrastructure WLAN, there are two kinds of ICMP message type 13 packets [4]. A router is able to generate the ICMP timestamp request packet which contains the value of its system timer when it sends the packet. Once receiving the request packet, a terminal responds with an ICMP timestamp reply packet that embraces two timestamps: the timestamp in the received request packet and that when the terminal sends the reply packet. For sake of simplicity, we name these two kinds of timestamps probe timestamps and response timestamps, respectively. Both these two types of timestamps do not get affected by the random medium access delays of the wireless medium as hardware sets the timestamp just before real transmission. In addition, the timer of a terminal is initialized at the time of its bootstrapping and incremented once every millisecond.

Our solution uses these two timestamps in response packets to estimate the device-to-device clock skew of a terminal with respect to the router and employs the clock skew as the terminal's fingerprint. A terminal's clock consists of primarily two parts: oscillator and counter. The oscillator is under the control of a crystal and ticks at a pre-fixed frequency. The counter is used to track the number of ticks generated by the oscillator. The exact frequency of a crystal is mainly determined by the the type of the crystal and the shape that it was cut relative to its axes. Nevertheless, due to the error of mechanic work, two crystals will have slightly different frequencies even they have the same type and the same cut [14]. Thus, the clock skews can be used to fingerprint wireless devices even they are made with similar clocks and in same environment.

Assume that the router has sent $n$ ICMP probe packets to a terminal and get $n$ response packets. Let the timestamp in the $i$-th response packet be $T_i$ and let $t_i$ be the timestamp in the probe packet. Let $S_i$ be the size of the $i$-th response packet and $R_i$ be the data rate at which $i$-th response packet is sent. Hence, the time that the router receives the $i$-th response packet is $T_i + S_i/R_i$, based on the terminal's clock. The difference between clocks of the router and terminal at the $i$-th response packet and probe packet can be calculated as $T_i + S_i/R_i - t_i$. Comparing with the clock difference at the first packet, we define the offset for the $i$-th response packet as $o_i$. Then we can get

$$o_i = (T_i + S_i/R_i - t_i) - (T_1 + S_1/R_1 - t_1)$$

In most of the cases, the data rate at the terminal remains fixed and the size of response packets are fixed as well [15]. Hence, $S_i/R_i = S_1/R_1$ and we can get

$$o_i = (T_i - T_1) - (t_i - t_1)$$

Let $x_i = t_i - t_1$ be the time difference between the first received packet and the $i$-th packet at the router. If we plot $(x_i, o_i)$, we can find the transition of the clock difference between the router and the terminal. Moreover, if the clock skew of a particular terminal remains constant, it can be estimated as the slope of this linear pattern. We call the set of $\{(x_1, o_1), \ldots, (x_n, o_n)\}$ the clock offset-set of the terminal.

## IV. FastID Design

### A. Overview

The system architecture of FastID consists of four components: *Data collector*, *Preprocessor*, *Clock skew calculator* and *Decision maker*. The data collector employs the probe packets in ICMP to fetch timestamps of terminals in replied response packets, then it outputs the clock offset-set to the preprocessor. The clock offset-set cannot be directly applied to estimate the clock skew as the packet sending on terminals is not stable and can be affected by some unknown delays, for example, the OS may unpredictably delay the packet sending at the network driver. Based on our observation, a few outliers of clock offsets can cause huge errors on clock skew calculation and fail the terminal identification. Therefore, the preprocessor is designed to filter those outliers based on the distribution of clock offsets. The clock skew calculator is used to estimate the clock skew from the filtered offset-set with a Linear Programming Method

(LPM). The Decision maker identifies the terminal that is being interrogated by the router according to its clock skew. To improve the accuracy of clock skew estimation, we design a novel algorithm of identification based on the similarity of distributions between the computed clock skew and those pre-registered on the router, instead of simply comparing their values of clock skews.

### B. Data Collector

When a terminal is signing into the network via a WiFi router, the router firstly suspends the request of connection, and then sends an amount of ICMP probe packets to the terminal. In this stage, we should consider two parameters which determine the overhead of FastID: the number of probe packets and the time interval between two consecutive probe packets. In FastID, we set the number of probe packets and time interval as 5,000 and 1 millisecond respectively based on our experimental results. The specific description of parameter selection is presented in Section VI-B.

### C. Preprocessor

The function of the preprocessor is to remove outliers in the clock offset-set. Outliers may be caused by the status of OS on the terminal, the environment or some other unknown reasons. Based on our observation, a small number of outliers can significantly degrade the accuracy of the skew computation.

To address this problem, FastID filters outliers based on the distribution of the clock offset-set. Specifically, every time that the router gets the ICMP response packet is independent of others, so the offset-set follows the normal distribution. Compute the mean value $Mean$ and the standard deviation $Std$ of the set, and then filter those offsets which are apart from $Mean$ more than $2Std$. The reason that we select $2Std$ as the threshold of filtering is: based on our experimental results, more than 20% to 30% offsets would be filtered if the threshold is set to the value of standard deviation. That makes the result of the skew calculation become unstable due to insufficient offsets. In contrast, some outliers could not be filtered and the accuracy of clock skew calculation will be degraded, if we choose the threshold more than $2Std$. Repeat this preprocessing iteratively for a couple of times to further refine the offset-set.

Figure 2 illustrates an example of the effect of the preprocessing. From this figure, we can see that the measured clock skew is negative (-64 part per million, ppm) without preprocessing due to outliers. With preprocessing, FastID filters most outliers and makes the measured clock skew become its real value (42ppm). It should be noted that we use the measure parts per million, essentially $\mu$s/s, denoted as ppm, to quantify clock skew.

### D. Clock Skew Calculator

We use the Linear Programming Method, LPM, to estimate the clock skew of a terminal from its clock offset-set. Given an offset-set $\{(x_1, o_1), \ldots, (x_n, o_n)\}$, LPM finds a line $\delta x+\phi$, where $\delta$ is the slope of the line and $\phi$ is the y-axis intercept,

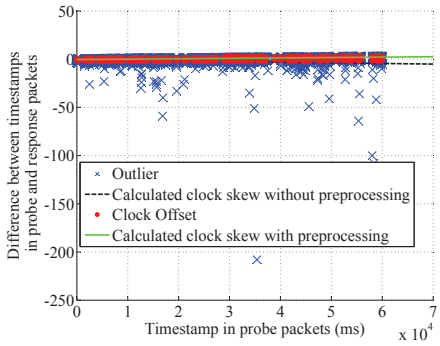



Fig. 2: An example of the effect of the preprocessing. A few outliers make the calculated clock skew (-64ppm) deviate from the real skew (42ppm).

that upper bounds the points in the clock offset-set of the terminal and outputs the slope of the line as the estimated clock skew. Thus, the clock skew estimation $\delta$ is such that any $i = 1, \ldots, n$, $\delta x_i + \phi \geq o_i$, and the following function is minimized:

$$1/n \sum_{i=1}^{n} (\delta x_i + \phi - o_i)$$

We list the calculated clock skews of ten terminals used in our experiment in Table I. Also, we can use Least-Square Fitting, LSF, to estimate the clock skew of a terminal from its clock offset-set. Given an offset-set $\{(x_1, o_1), \ldots, (x_n, o_n)\}$, LSF searches a line $\delta x + \phi$ such that

$$\sum_{i=1}^{n} (o_i - (\delta x_i + \phi))^2$$

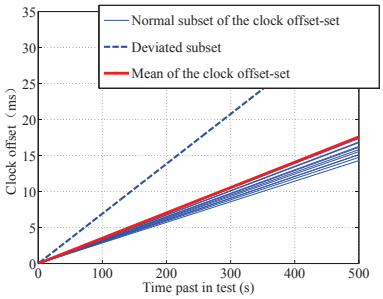
remains minimum. The slope of the line $\delta$ is estimated as the clock skew of the clock offset-set.

One of the main differences of LSF from LPM is its lack of tolerance toward outliers. A few outliers can cause the clock skew estimated by LSF varying significantly from the clock skew determined by the majority of the points. This incurs problems while estimating clock skew from data with outliers. Therefore, in FastID, we use the LPM for clock skew estimation.

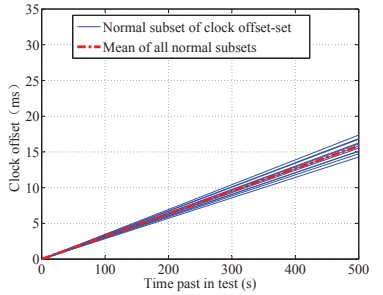
### E. Decision Maker

The decision maker is used to separate the clock offset-sets and then identify corresponding terminals. Basically, the router can identify terminals directly according to their calculated clock skews. That is, the decision space to differentiate a terminal's clock skew could be simply represented with its mean and standard deviation. However, this basic method is not resilient to overlapped decision spaces if two terminals have close clock skews (e.g. terminal 2 and 3 in Table I), and then, cannot achieve high accuracy of identification. Therefore, we need other techniques to address this problem.

Fitting multiple lines to a data set is not a new problem. Generalized Hough Transform (GHT) [16], [17] and Randomized Hough Transform (RHT) [18] are mature techniques for this purpose in the field of computer vision. However, they




(a) A subset deviates from others and makes the calculated skew vary from the real one.




(b) FastID removes the deviated subset and improves the accuracy of the skew calculation.

Fig. 3: An example that FastID removes the deviated subset.

are computationally intensive and require a large amount of storage. Another approach to solve this problem is to model the data and employ the statistical method of expectation maximization (EM) to separate the data [19]. However, the EM algorithm should select the initial parameters very carefully and the results heavily depends on the values of these initial parameters.

In our FastID, we present a lightweight statistic method based on the observation from our experiments: the mean of an offset-set of a terminal in each measurement of clock skews remains nearly same but the standard deviation varies due to delays incurred by some reasons such as the environment and the status of the OS of the terminal. Borrowing ideas from the statistical method, we design the identification algorithm as shown in Alg. 1.

As described in Section III-B, the distribution of the clock offset-set of a terminal follows the normal distribution, and the mean is the clock skew. Let distributions of $n$ registered terminals (i.e. the router allows these $n$ terminals to access the network.) be $\mathcal{N}(Mean^{(1)}, Std^{(1)}), \ldots, \mathcal{N}(Mean^{(n)}, Std^{(n)})$. Where $Mean^{(j)}$ and $Std^{(j)}$ denote the mean and the standard deviation of the registered terminal $j$, respectively. Thus, the decision space for the terminal $j$ can be $[Mean^{(j)} - Std^{(j)}, Mean^{(j)} + Std^{(j)}]$. When an unidentified terminal is signing into the router, the distribution of the clock skew of the terminal can be computed and represented as $\mathcal{N}(Mean', Std')$. The router searches all registered terminals and finds $l$ registered terminals which have similar skews to the unidentified terminal. We call these terminals as *similar terminals*. That is, for a similar terminal $j$, $|Mean^{(j)} - Mean'| \leq Std'$. If no similar terminal exists, the router rejects the connection request of this unidentified terminal. Otherwise, the router further identifies the terminal as following.

**Algorithm 1** Terminal identification

```
Compute the distribution of unidentified terminal N(Mean', Std');
for each registered terminal i do
    if |Mean^(i) − Mean'| < Std' then
        Denote the registered terminal i as a similar terminal;
    end if
end for
if There is no similar terminal then
    Reject the connection request of the unidentified terminal;
    Return;
end if
for Each similar terminal j do
    Divide the offset-set of the unidentified terminal into m subsets G_1, ..., G_m;
    for Each subset G_i do
        Compute the distribution N(Mean_i, Std_i);
        if Pr[Mean_i] > Pr[Mean^(j) − Std^(j)] then
            Mark the ith bit of V_j as 1;
        else
            Mark the ith bit of V_j as 0;
        end if
    end for
    Get the similarity vector V_j;
end for
Combine all similarity vectors to be the similarity matrix M;
Find the row with maximum Hamming Weight in M;
Recognize the unidentified terminal as the corresponding similar one.
```

Based on our observation, $Std'$ is much larger than standard deviations of similar terminals as the environment and the current status of the unidentified terminal may give some delays on packet sending. Therefore, the decision spaces of those similar terminals may be overlapped to some extent, so that we cannot directly recognize the unidentified terminal as a similar one which has the closest mean. Let the unidentified terminal be the similar one $j$. We can construct a new distribution as $\mathcal{N}(Mean^{(j)}, Std')$. To compute the similarity between the unidentified terminal and the similar terminal $j$, we divide the whole clock offset-set of the unidentified terminal into $m$ non-overlapped and consecutive subsets $G_1, \ldots, G_m$, and count every distribution $\mathcal{N}(Mean_i, Std'_i)$ of $G_i$.

The router firstly computes the probability $Pr[Mean_i]$ in the distribution of $\mathcal{N}(Mean^{(j)}, Std')$, and then compares $Pr[Mean_i]$ with the probability of the threshold $Pr[Mean^{(j)} - Std^{(j)}]$ (or $Pr[Mean^{(j)} + Std^{(j)}]$). If $Pr[Mean_i] > Pr[Mean^{(j)} - Std^{(j)}]$, it means that the $G_i$ follows the distribution of the similar terminal $j$ and the router

TABLE I: Terminal list in our experiments

| Terminal ID | Operating System | Type Number | Terminal Type | Measured Clock Skew |
|---|---|---|---|---|
| 1 | Ubuntu 12.04 and Windows 8.1 | AcerE1-471G | Laptop | 46.08±0.80 |
| 2 | Ubuntu 12.04 and Windows 8.1 | AcerE1-471G | Laptop | 41.623±0.59 |
| 3 | Android 4.1 | Teclast P88 | Tablet | 41.94±2.68 |
| 4 | Ubuntu 12.04 and Windows 8.1 | Acer Aspire 4755G | Laptop | 31.65±1.76 |
| 5 | Android 4.1.2 | Samsung GT-I9308 | Smartphone | 27.20±2.81 |
| 6 | Ubuntu 12.04 and Windows 8.1 | Asus R510VC | Laptop | 54.53±1.78 |
| 7 | Android 3.4 | Mi 1S | Smartphone | 47.43±3.12 |
| 8 | Android 2.3.8 | Motorola ME525 | Smartphone | -4.25±4.20 |
| 9 | Android 4.1 | Sony SGPT111CN/S | Tablet | 11.48±2.68 |
| 10 | Android 4.1 | Teclast P88 | Tablet | 44.86±1.33 |

marks the $i$-th bit $V_j[i]$ of a vector $V_j$ as 1 to record the result. Where $V_j$ is a $m$-bit long vector to represent the similarity of the unidentified terminal to the similar terminal $j$. Otherwise, $V_j[i]$ is marked as 0. Repeating this comparing procedure for each subset, we can get a similarity vector $V_j$ in terms of the similar terminal $j$. Figure 3 gives an example that FastID removes the subset that does not follow the distribution of the whole clock offset-set. From Fig.3(a),we can see that there is a subset of the clock offset-set which deviates from the others, so that the calculated clock skew also deviates from the real skew. With our algorithm, FastID can effectively remove those deviated subsets and makes the calculation of clock skew more accurate, as shown in Fig. 3(b).

Repeat the procedure above for each similar terminal, we can get a $l \times m$ matrix $M$ of similarity by combining all similarity vectors $(V_1, \ldots, V_j, \ldots, V_m)$. In this matrix, the Hamming weight of each row represents the similarity of the corresponding similar terminal to the unidentified one. Thus, the router recognizes the unidentified terminal as the similar terminal that corresponds to the row with the maximum Hamming Weight in the matrix.

## V. FastID Implementation

We implement FastID presented in the last section on a MERCURY MW150R WiFi router running OpenWRT Attitude Adjustment Release (12.09) [5]. OpenWRT is a Linux distribution for embedded devices. It provides a fully writable filesystem with package management and allows users to customize the device through the use of packages to suit any application. The MERCURY router is equipped with Atheros AR9331 chipset that works with the MadWifi driver. We choose this router because it's an off-the-shelf product and can be supported by OpenWRT.

In order to timestamp the ICMP probe packets with high resolution, we implement our approach in OpenWRT and then burn the new firmware into the router. Specifically, FastID firstly extracts the value of the system clock in millisecond on board, and then translates the value to the format of ICMP timestamp. FastID employs the Raw Socket [20] to generate the probe packets. Where the Raw Socket is an internet socket that allows direct sending and receiving of Internet Protocol packets without any protocol-specific transport layer formatting.

In addition, there are two methods to control the access of a terminal to the internet before it has been identified. First, the router establishes connection between the terminal and the internet only after the terminal being identified as valid. However, it is very difficult to be implemented as we have to modify the kernel of the OpenWRT. Thus, we prefer the second method: the router firstly lets a terminal connect to the internet but blocks any traffic from the terminal till identified as valid.

## VI. Experimental Results

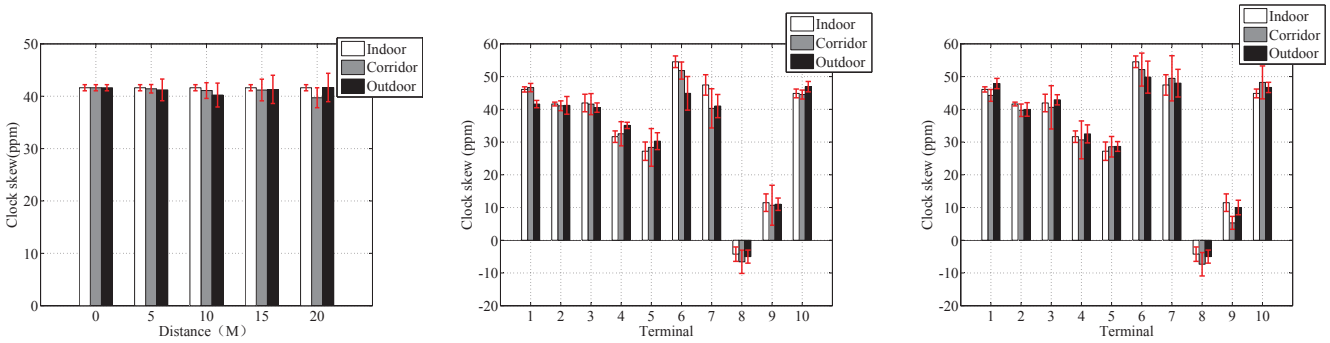
In this section, we evaluate FastID based on a set of 10 commodity WiFi terminals, including four laptops, three smartphones and three tablets. Among these terminals, four laptops are installed both Windows 8.1 and Ubuntu 12.04. Tablets and smartphones are installed Android OS, as shown in Table. I.

We test FastID in three environments which are most common for WiFi applications: indoor, outdoor and corridor. Also, we evaluate the effectiveness of FastID under different distance between terminals and the router. For each experiment, we repeat testing for 1,000 times. Our evaluation focuses on three aspects of FastID performance including: (1) The efficacy of identifying terminals by clock skews; (2) The overhead and (3) accuracy of FastID.

### A. Efficacy of Identification

**Identification range**: We evaluate the identification range of FastID in three environments: outdoor, indoor and corridor. The results are plotted in Fig. 4. From the Fig. 4(a), we can see that the clock skew of the terminal 2 remains nearly same in different environments as well as distances between the router and the terminal. Figure 4(b) and 4(c) show that the change of clock skews is not significant in 10 and 20 meters, respectively. We measure the clock skews at most 20 meters distance as the transmission power of the router can only support this range, but we should note that this range satisfies real applications in most cases.

**Impact of different operating systems**: The whole TCP/IP protocol stack including the ICMP protocol is implemented in the kernel of OS. As a result, unproper implementations of the TCP/IP protocol stack in operating systems may influence the ICMP packets processing and incur errors in the clock skew calculation. To this end, we measure clock skews of four laptops which are installed both Linux Ubuntu 12.04 and Windows 8.1. The result is plotted in Fig.5. From this figure, we can see that FastID makes measured clock skews change very slightly at the same terminal with out preprocessing and identification algorithms.




(a) Clock skew of terminal 2 in different distances and environments.

(b) Clock skews of all terminals at 10 meters away from the router.

(c) Clock skews of all terminals at 20 meters away from the router.

Fig. 4: Clock skews measured in different environments and distances.

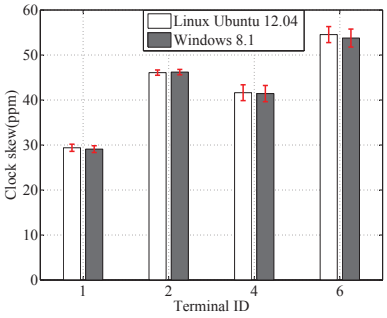



Fig. 5: Measured clock skews in different operating systems.

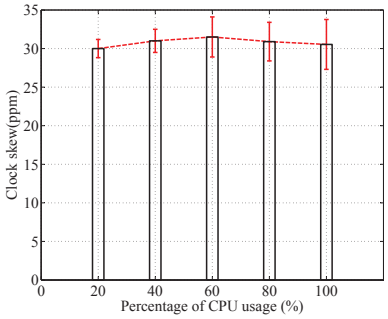



Fig. 6: Measured clock skews VS percentages of CPU usage.

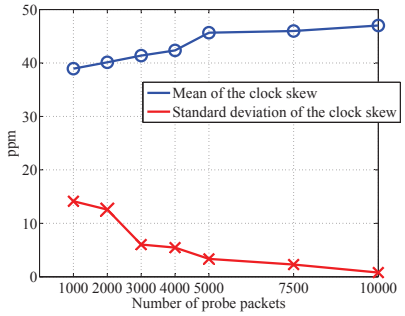



Fig. 7: Mean and standard deviation of calculated clock skews VS the number of probe packets.

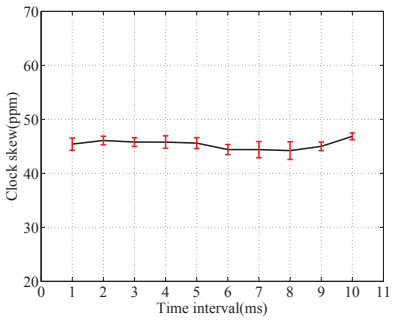



Fig. 8: Measured clock skew VS the time interval between two consecutive probe packets.

**Impact of running status of operating systems**: In FastID, ICMP response packets are automatically generated by the operating system. Thus, the running status of the OS may bring some delays to the processing of ICMP packets. For example, the OS of a terminal may postpone the executing of network processes if the CPU is busy on some computation intensive tasks. As a result, unpredictable delays may be involved in the timestamps of ICMP response packets on the terminal and fail the clock skew computation on the router. Therefore, we design an identification algorithm described in Alg. 1 based on the similarity matrix (see Section IV-E) to minimize the impact of those unpredictable delays. To verify the effectiveness of this algorithm, we conduct an experiment to prove that clock skews would not be influenced much by the status of operating systems in our approach. In this experiment, we use the percentage of CPU usage to depict the status of operating systems and then measure clock skews. The result is plotted in Fig. 6. From this figure, we can see that the usage of CPU affects the value of clock skews very little and the terminal can be still identified even its percentage of CPU usage is 100%.

### *B. Overhead of FastID*

We now evaluate the overhead of FastID. As described in Section IV-B, there are two major factors that determine the overhead on the measurement delay and computation cost: the number of probe packets and the time interval between two consecutive probe packets.

**Number of probe packets**:Obliviously, more probe packets lead to higher accuracy of clock skew calculation. However, a large number of probe packets also cause unacceptable measurement delay and computation cost. Therefore, we should leverage the trade-off between the number of probe packets

and accuracy of clock skew calculation and choose a proper value which is practical in real-life applications. Figure 7 depicts the relationship between the number of probe packets and the clock skew computation. From this figure, we can find that the mean becomes stable when the router have sent 5,000 probe packets, and the standard deviation also decreases to a relative low value which means the router can identify terminals accurately.

**Time interval**:We set the time interval between two consecutive probe packets from 1 to 10 milliseconds and plot the result of clock skew in Fig. 8. This figure shows that the change of clock skew is statistically not significant, so we can use the minimum time interval, 1 ms, for fast probe packets sending. By combining the results of the time interval and number of probe packets, we can find that the least measurement delay of FastID is about $1ms \times 5,000 = 5$ seconds. We also measure the delay of clock skew computation by setting a timer in the router to count the time consumption of computation. The average computation delay among 1,000 computations is less than 0.1 ms. It means that FastID can efficiently identify a terminal in less than 6 seconds.

### C. Accuracy

To evaluate the accuracy of FastID, we use two concepts to represent the false rate: False Accept Rate (FAR) and False Reject Rate (FRR). FAR is the probability that FastID falsely accepts an invalid terminal as a registered one, while FRR means the probability that FastID rejects a registered terminal in error. Through our experiments, we find that the FRR is always zero under different settings, which means FastID always accepts valid terminals, and the FAR is about 1% in the condition of proper parameters. Part of results are listed in Table II. In this table, we set the numbers of clock skew measurements, clock skew measurements per terminal and offsets per measurement are 1,000, 100 and 5,000 respectively.

Repeating our experiments for 1,000 times, we can see that clock skews remain consistent over time for the same terminal but vary significantly across terminals. By filtering outliers and identifying terminals with the similarity matrix, FastID is able to compute clock skews with small standard deviations which are very suitable for fingerprinting WiFi terminals. The result of experiments on different operating systems presents that clock skews are independent of operating systems if we can remove noise data caused by OS. FastID can provide effective terminal identification in most common environments of WiFi applications. Moreover, FastID has best performance in indoor scenario as the multi-path effect of RF can provide more reliable communication channel for packet sending and receiving. The results of overhead illustrate that FastID can efficiently identify a terminal within 6 seconds which is very suitable for current terminal login applications.

TABLE II: Accuracy of FastID

| Time interval (ms) | # of subsets in decision | Accuracy(%) | FAR(%) |
|---|---|---|---|
| 1 | 10 | 96% | 4% |
| 1 | 15 | 98% | 2% |
| 1 | 20 | 99% | 1% |
| 2 | 10 | 97% | 3% |
| 2 | 20 | 99% | 1% |
| 4 | 20 | 99% | 1% |
| 6 | 20 | 99% | 1% |
| 8 | 20 | 99% | 1% |
| 10 | 20 | 99% | 1% |

## VII. Discussions

### A. Timestamp Spoofing

Our approach to identify a wireless terminal is based on the clock skew of the terminal. As probe and response packets are broadcasted by the router and terminal, an attacker can listen to these packets, and then compute the relative clock skew of the terminal with respect to the router's. To defend against this attack, FastID can set the router not to embed timestamps of packet sending in probe packets. By which means, the attacker can only infer the relative clock skew of the the terminal with respect to its own. With this clock skew estimate, the attacker may be able to impersonate the terminal by spoofing timestamps by adding its own timestamps with proper offsets.

The timestamp spoofing is conceptually feasible, but infeasible in practice. The reason is: in the 802.11 wireless network medium access control, the sender has to sense channel status before sending any packet. If the channel is detected as idle for the Distributed Inter frame Sequence (DIFS) duration, the sender would randomly delay its transmission by a number of time slots. The length of every time slot is chosen from the interval [0, *CW*] at random, where *CW* is the contention window size. The exact time between when the wireless network driver hands over a wireless packet and when the packet is actually sent is unpredictable due to these random delays. As a result, the timestamp forged by the attacker cannot reflect the real time of transmission.

### B. Effect of NTP Synchronization of Routers Clock on Skew Estimate

Clocks of routers that are connecting to a network are synchronized through the Network Time Protocol (NTP) or any other clock synchronization mechanism. In our approach, we measure clock skews of wireless terminals relative to the router. Therefore, clock skews of same terminals may vary significantly once the clock of the router is synchronized to clocks of other routers in the network, so that the router has to re-calculate clock skews of terminals. However, our approach can still work in this case. As aforementioned in Section VI-B, our measurement time is small enough (less than 6 seconds) and we measure clock skews in less than millisecond. The accuracy of NTPv4 is within 10 milliseconds over the internet and within 200 microseconds over an LAN. The default minimum polling interval of NTP is 64 seconds [21]. Thus, in our approach, the router enabled NTPv4 can still calculate clock skews of terminals with enough accuracy. Furthermore, we can leverage the NTP polling interval of the router to a longer value, e.g. 10 minutes or longer, to reduce the overhead of clock skew re-calculation.

### C. Clock Skew Measurement with TCP Timestamps

Our approach can also use the TCP timestamps [22] to fingerprint WiFi terminals with the same methodology. The drawback of using the TCP protocol is that we have implement a client and install it on terminals to reply the TCP probe packets from the router, as there is no automatic mechanism of packet response in TCP protocol. We thereby just implement our approach based on TCP protocol for the purpose of comparison. The result is plotted in Fig.9. From this figure, we can see that there is no remarkable difference on the measurement of clock skews between TCP and ICMP protocols. It means that our approach is flexible to network protocols and can be applied in the network in which the ICMP protocol is unavailable due to security concern.

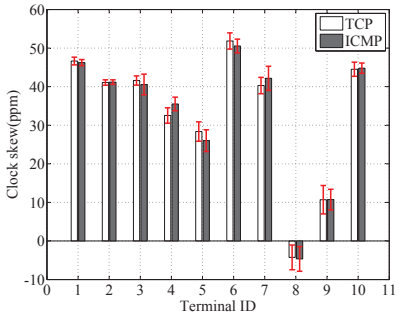



Fig. 9: Measured clock skews with both ICMP and TCP protocols.

### D. Operating Systems that Support Our Approach

Currently, our approach is supported by Linux, Android and Windows 8. Regarding Windows 7, there is a bug in its implementation of ICMP protocol that the accuracy of timestamps is in second, which makes FastID is not compliant to Windows 7. However, Microsoft is releasing a patch for this bug in 2015, thus our approach is still available for Windows 7 in near future.

### E. Impact of Channel Load

Load on WiFi channels could affect the packet sending rate of WiFi routers. Like, heavy workload may slow down the packet sending rate and then increase the time interval between two probe packets sendings of FastID. In our experiments, we do not exam the impact of channel load. However, from Fig. 8 and Tab II, we can see that time interval makes no difference on the accuracy of FastID. The only impact of channel load on FastID is the time delay of identification. We will do experiments to exam the performance of FastID under different channel workloads in future for the journal version.

## VIII. Conclusion

In this paper, we present FastID, a fast and accurate system that identifies 802.11 wireless terminals using the clock skews of their chipsets. Previous clock skew based approaches can only passively listen to fingerprintees and extract clock skews after long time (maybe hours or tens of minutes) of data collection. In comparison, in FastID the router can actively send probe ICMP packets to fetch timestamps of wireless terminals, such that FastID can identify terminals within several seconds, which significantly enhance the security of WLAN against unauthorized access to wireless routers. To implement FastID, we develop efficient algorithms, including outlier filtering and identifying terminals based on the similarity of their clock skew distributions, to achieve highly accurate fingerprinting and identification. We realize FastID on an off-the-shelf commercial WiFi router. Through extensive evaluations under common used application settings, we demonstrate that FastID enables efficient and effective terminal identification with high accuracy, short delay and low cost.

## Acknowledgement

We would like to thank anonymous reviewers for their insightful comments. This work is supported by National Natural Science Foundation of China (Grant No. 61472068, 61173171), and the Project funded by China Postdoctoral Science Foundation (Grant No. 2014M550466).

## References

[1] J. Huang et al. Blueid: A practical system for bluetooth device identification. In *IEEE Infocom*, 2014.
[2] T. Kohno et al. Remote physical device fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 2(2), 2005.
[3] S. Jana and S.K. Kasera. On fast and accurate detection of unauthorized wireless access points using clock skews. *IEEE Transactions on Mobile Computing*, 9(3), 2010.
[4] J. Postel et al. Rfc 792: Internet control message protocol. *InterNet Network Working Group*, 1981.
[5] Openwrt wireless freedom, https://openwrt.org.
[6] J. Pang et al. 802.11 user fingerprinting. In *ACM MobiCom*, 2007.
[7] Kount complete - product overview, http://www.kount.com/products/kount-complete.
[8] F. Jason et al. Passive data link layer 802.11 wireless device driver fingerprinting. In *Usenix Security*, 2006.
[9] R. Zhou et al. Wiznet: A zigbee-based sensor system for distributed wireless lan performance monitoring. In *IEEE PerCom*, 2013.
[10] M. Mariyam et al. Fingerprinting 802.11 rate adaption algorithms. In *IEEE Infocom*, 2011.
[11] B. Vladimir et al. Wireless device identification with radiometric signatures. In *ACM MobiCom*, 2008.
[12] B. Danev and S. Capkun. Transient-based identification of wireless sensor nodes. In *ACM/IEEE IPSN*, 2009.
[13] K. Mustafa. Defining skew, propagation-delay, phase offset (phase error). txas instruments application report - scaa055. Technical report, 2001.
[14] IEEE Standards Coordinating Committee. Ieee guide for measurement of environmental sensitivities of standard frequency generators, 27-scc27-on time and frequency.
[15] IEEE Computer Society LAN MAN Standards Committee. Wireless lan medium access control (mac) and physical layer (phy) specifications, 1997.
[16] P.C. Hough. Method and means for recognizing complex patterns, us patent 3069654, 1962.
[17] D.H. Ballard. Generalizing the hough transform to detect arbitrary shapes. *Pattern Recognition*, 13(2), 1981.
[18] X. Lei and O. Erkki. Randomized hough transform (rht): basic mechanisms, algorithms, and computational complexities. *CVGIP: Image understanding*, 57(2), 1993.
[19] A.P. Dempster et al. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 1977.
[20] Raw socket, http://en.wikipedia.org/wiki/Raw_socket.
[21] D. L. Mills. Network time protocol version 4 reference and implementation guide. *Electrical and Computer Engineering Technical Report*, 2006.
[22] V. Jacobson et al. Rfc 1323: Tcp extensions for high performance, 1992.